Integrate with PostgreSQL

This tutorial demonstrates a simple workflow to integrate ksqlDB with an instance of PostgreSQL.

Prerequisites:

- Confluent Platform [https://docs.confluent.io/current/installation/installing_cp/index.html] is installed an running. This installation includes a Kafka broker, ksqlDB, ZooKeeper, Schema Registry and Connect.
- If you installed Confluent Platform via TAR or ZIP, navigate into the installation directory. The paths and commands used throughout this tutorial assume that you are in this installation directory.
- Consider installing [https://docs.confluent.io/current/cli/installing.html] the Confluent CLI to start a local installation of Confluent Platform.
- Java: Minimum version 1.8. Install Oracle Java JRE or JDK >= 1.8 on your local machine

Installing JDBC Source Connector Plugin

If you installed Kafka Connect via Confluent Platform, then it comes with an installation of the JDBC source connector. Otherwise, install it from Confluent Hub.

Installing Postgres via Docker

If you are experimenting with the ksqlDB-Connect integration and don't have a PostgreSQL instance locally, you can install it by using Docker and populate some data:

Install PostgreSQL by using the `docker pull postgres` command. Start the database and expose the JDBC port:

```
docker run -p 5432:5432 --name some-postgres -e POSTGRES_USER=$USER -e
POSTGRES_DB=$USER -d postgres
```

Run PSQL to generate some data:

```
docker exec -it some-postgres psql -U $USER
psql (11.5 (Debian 11.5-1.pgdg90+1))
Type "help" for help.

postgres=# CREATE TABLE users (username VARCHAR, popularity INT);
CREATE TABLE
postgres=# INSERT INTO users (username, popularity) VALUES ('user1',
100);
INSERT 0 1
postgres=# INSERT INTO users (username, popularity) VALUES ('user2',
5);
INSERT 0 1
postgres=# INSERT INTO users (username, popularity) VALUES ('user3',
75);
INSERT 0 1
```

When you're done, clear your local state by using the `docker kill` command.

```
docker kill some-postgres && docker rm some-postgres
```

Create a JDBC Source Connector

Now that Postgres is up and running with a database for your user, you can connect to it via ksqlDB. If you're using the default configurations, ksqlDB connects automatically to your Connect cluster. Otherwise, you must change the `ksql.connect.url` property to point to your Connect deployment.

```
CREATE SOURCE CONNECTOR `jdbc-connector` WITH(
  "connector.class"='io.confluent.connect.jdbc.JdbcSourceConnector',
  "connection.url"='jdbc:postgresql://localhost:5432/YOUR_USERNAME',
  "mode"='bulk',
  "topic.prefix"='jdbc-',
  "key"='username');
```

Profit

At this point, data should automatically start flowing in from Postgres to ksqlDB. Confirm this by running the following statement.

```
DESCRIBE CONNECTOR "jdbc-connector";
```

Your output should resemble:

```
Name                 : jdbc-connector
Class                : io.confluent.connect.jdbc.JdbcSourceConnector
Type                 : source
State                : RUNNING
WorkerId             : 10.200.7.69:8083

 Task ID | State   | Error Trace
---------------------------------
 0       | RUNNING |
---------------------------------

 Related Topics
----------------
 jdbc-users
----------------
```

Import this topic as a table to ksqlDB by using the following command.

```
CREATE TABLE JDBC_USERS WITH(value_format='AVRO', kafka_topic='jdbc-
users');
```

Select everything from the topic to see how it gets auto populated:

```
SELECT * FROM JDBC_USERS EMIT CHANGES;
```

You output should resemble:

```
+-------------------+-------------------+-------------------+------------
------+
|ROWTIME            |ROWKEY             |USERNAME           |POPULARITY
|
+-------------------+-------------------+-------------------+------------
------+
|1566336783102      |user1              |user1              |100
|
|1566336783102      |user2              |user2              |5
|
|1566336783102      |user3              |user3              |75
|
|1566336788106      |user1              |user1              |100
|
|1566336788106      |user2              |user2              |5
|
|1566336788106      |user3              |user3              |75
|
```

Note that users are repeated multiple times. This means that `bulk` mode is specified, which re-imports the entire database every time. Obviously, this isn't appropriate for production. For more information on changelog capture, see Incremental Query Modes [https://docs.confluent.io/current/connect/kafka-connect-jdbc/source-connector/index.html#incremental-query-modes].

Page last revised on: 2019-12-17

Last update: 2019-12-17